# On Deftly Introducing Procedural Elements into Unification Parsing

**R. Bobrow**
**Lance Ramshaw**

BBN Systems and Technologies Corp.
10 Moulton Street, Mailstop 6/4C
Cambridge, MA 02138

## 1 Introduction

Unification grammars based on complex feature structures are theoretically well-founded, and their declarative nature facilitates exploration of various parsing strategies. However, a straightfoward implementation of such parsers can be painfully inefficient, exploding lists of possibilities and failing to take advantage of search control methods long utilized in more procedurally-oriented parsers. In the context of BBN's Delphi NL system, we have explored modifications that gain procedural efficiency without sacrificing the theoretical advantages of unification-based CFG's.

One class of changes was to introduce varieties of structure sharing or "folding" to control combinatorics. One kind of sharing was achieved automatically by partially combining similar grammar rules in the tables used by the parser. Another resulted from introducing a strictly limited form of disjunction that grammar writers could use to reduce the number of separate rules in the grammar.

The other class of changes introduced procedural elements into the parsing algorithm to increase execution speed. The major change here was adding partial prediction based on a procedurally-tractable and linguistically-motivated subset of grammar features. Appropriate choice of the features on which to base the prediction allowed it to cut down substantially the space that needed to be searched at runtime. We are also exploring the use of non-unification computation techniques for certain subtasks, where the nature of the computation is such that approaches other than unification can be significantly faster but can still be integrated effectively into an overall unification framework.

Together, the classes of changes discussed here resulted in up to a 40-fold reduction in the amount of structure created by the parser for certain sentences and an average 5-fold parse time speedup in the BBN Delphi system. We believe that they represent significant new directions for making the theoretically elegant unification framework also an efficient environment for practical parses.

## 2 Folding of Similar Structures

Major improvements were obtained by partially combining similar structures, either automatically, by combining common elements of rules, or manually, through the use of a limited form of disjunction.

### 2.1 Automatic Rule Folding

The goal here is to provide an automatic process that can take advantage of the large degree of similarity frequently found between different rules in a unification grammar by overlapping their storage or execution paths. While the modularity and declarative nature of such a grammar are well-served by representing each of a family of related possibilities with its own rule, storing and testing and instantiating each rule separately can be quite expensive. If common rule segments could be automatically identified, they could be partially merged, reducing both the storage space for them and the computational cost of matching against them.

We have implemented a first step toward this sort of automatic rule folding, a scheme that combines rules with equivalent first elements on their right hand sides into *rule-groups*. This kind of equivalence between two rules is tested by taking the first element of the right hand side of each rule and seeing if they mutually subsume each other, meaning that they have the same information content. If so, the variables in the two rules are renamed so that the variables in the first elements on their right hand sides are named the same, meaning that those elements in the two rules become identical, although the rest of the right hand sides and the left hand sides may still be different. This equivalence relation is used to divide the rules into equivalence classes with identical first right hand side elements.

Before this scheme was adopted, the parser, working bottom up and having discovered a consituent of a particular type covering a particular substring, would individually test each rule whose right hand side began with a constituent of that category to see if this element could be the first one in a larger constituent using that rule. After adding rule-groups, the parser only needs to match against the common first right side element for each group. If that unification fails, none of the rules in the group are applicable; if it succeeds, the resulting substitution list can be used to set up continuing configurations for each of the rules in the group. Use of the rule-groups scheme collapsed a total of 1411 rules into 631 rule-groups, meaning that each single test of a rule-group on the average did the work of testing between two and three individual rules.

The success of this first effort suggests the utility of further compilation methods, including folding based on right side elements beyond the first, or on shared tails of rules, or

shared central subsequences. Note that the kind of efficiency added by this use of rule-groups closely resembles that found in ATN's, which can represent the common prefixes of many parse paths in a single state, with the paths only diverging when different tests or actions are required. But because this process here occurs as a compilation step, it can be added without losing any of unification's modularity and clarity. While similar goals could also be achieved by rewriting the grammar, for example, by creating a new constituent type to represent the shared element, such changes would make the grammar less perspicuous. The grammar writer should be free to follow linguistic motivations in determining constituent structure and the like, and let the system take care of restructuring the grammar to allow it to be executed more efficiently.

## 2.2 Limited Disjunction

While in some circustances, it seems best to write independent rules and allow the system to discover the common elements, there are other cases where it is better for the grammar writer to be able to collapse similar rules using disjunction. That explicit kind of disjunction, of course, has the same obvious advantage of allowing a single rule to express what otherwise would take $n$ separate rules which would have to be matched against separately and which could add an additional factor of $n$ ambiguity to all the structures built by the parser. For example, the agreement features for a verb like "list" used to be expressed in BBN's Delphi system using five separate rules:

```
(V (AGR (1ST) (SNG)) ...) → (list)
(V (AGR (2ND) (SNG)) ...) → (list)
(V (AGR (1ST) (PL)) ...) → (list)
(V (AGR (2ND) (PL)) ...) → (list)
(V (AGR (3RD) (PL)) ...) → (list)
```

That information can be expressed in a single disjunctive rule:

```
(V (:OR (AGR (1ST) (SNG))
        (AGR (2ND) (SNG))
        (AGR (1ST) (PL))
        (AGR (2ND) (PL))
        (AGR (3RD) (PL))) ...) → (list)
```

Many researchers have explored adding disjunction to unification, either for grammar compactness or for the sake of increased efficiency at parse time. The former goal can be met as in Definite Clause Grammars [6] by allowing disjunction in the grammar formalism but multiplying such rules out into disjunctive normal form at parse time. However, making use of disjunction at parse time can make the unification algorithm significantly more complex. In Karttunen's scheme [4] for PATR-II, the result of a unification involving a disjunction includes "constraints" that must be carried along and tested after each further unification, to be sure that at least one of the disjuncts is still possible, and to remove any others that have become impossible. Kasper [5], while showing that the consistency problem for disjunctive descriptions is NP-complete, proposed an approach whose average complexity is controlled by separating out the disjunctive elements and postponing their expansion or unification as long as possible.

Rather than pursuing efficient techniques for handling full disjunction within unification, we have taken quite a different tack, defining a very limited form of disjunction that can be implemented without substantially complicating the normal unification algorithm. The advantage of this approach is that we already know that it can be implemented without significant loss of efficiency, but the question is whether such a limited version of disjunction will turn out to be sufficiently powerful to encode the phenomena that seem to call for it. Our experience seems to suggest that it is.

Much of the complexity in unifying a structure against a disjunction arises only when more than one variable ends up being bound, so that dependencies between possible instantiations of the variables need to be remembered. For example, the result of the following unification

```
?AGR ⊓ (:OR (AGR (2ND) (SNG))
            (AGR (2ND) (PL))
            (AGR (3RD) (PL)))
```

(where "?" marks variables and "⊓" means unification) can easily be represented by a substitution list that binds ?AGR to the disjunction itself, but the following case

```
(AGR ?P ?N) ⊓ (:OR (AGR (2ND) (SNG))
                   (AGR (2ND) (PL))
                   (AGR (3RD) (PL)))
```

requires that the values given to ?P and ?N in the substitution list be linked in some way to record their interdependence. In particular, it seemed that if we never allowed variables to occur inside a disjunction nor any structure containing more than one variable to be matched against one, then the result of a unification would always be expressible by a single substitution list and that any disjunctions in that substitution list would also be only disjunctions of constants. Thus we required that disjuncts contain no variables, and that the value matched against a disjunction either be itself a variable or else contain no variables.

However, enforcing the restriction against unifying disjunctions with multi-variable terms turns out to be more complex than first appears. It is not sufficient to ensure, while writing the grammar, that any element in a rule that will directly unify with a disjunctive term be either a constant term or a single variable, since a single variable in the rule that directly matches the disjunctive element might have already, by the operation of other rules, become partially instantiated as a structure containing variables, and thus one that our limited disjuntion facility would not be able to handle.

For example, if the disjunctive agreement structure for "list" cited above occurs in a clause with a subject NP whose agreement is (AGR (3RD) (PL)), and in a containing VP rule that merely identifies the two values by binding them both to a single variable, the conditions for our constrained disjunction are met. However, if the subject NP turns out to be pronoun with its agreement represented as (AGR ?P ?N), the constraint is no longer met.

This problem with our fast but limited disjunction turned up, for example, when a change in a clause rule caused agreement features to be unbound at the point where disjunctive matching was encountered. The change was introduced to allow for queries like "Do United or American have flights ...", where the agreement between the conjoined subject and the verb does not follow the normal rules. The solution to that problem was to introduce a *constraint node* [1] pseudo-constituent, placed by convention at the end of the rule, to compute the permissible combinations of agreement values, with the values chained from the subject through this constraint node to the VP. Unfortunately, because of the placement of that constraint node in the rule, this meant that the agreement features were still unbound when the VP was reached, which caused our disjunctive unification to fail.

In our current implementation, the grammar writer who makes use of disjunction in a rule must also ensure that the combination of that rule with the existing grammar and the known parsing strategy will still maintain the property that any element to be unified with a disjunctive element will be either a constant or a single variable. Mistakes result in errors flagged during parsing when such a unification is attempted. We are not happy with this limitation, and are planning to expand the power of our disjunction mechanism using Kasper's methods [5] insofar as we can do so while maintaining efficiency. Nevertheless, the result of our work so far has been a many-fold reduction in the amount of structure generated by the parser without any significant increase in the complexity of the unification itself.

# 3 Procedural Algorithmic Elements

A second class of changes in addition to those that fold together similar structures are changes in the parsing algorithm that reduce the size of the search space that must be explored. The major type of such control was a form of prediction from left context in the sentence.

## 3.1 Prediction

A form of prediction for CFG's was described by Graham, Harrison, and Ruzzo [3] that was complete in the sense that, during its bottom-up, left-to-right parsing, their algorithm never tries to build derivations at word $w$ using rule $R$ unless there is a partial derivation of the left context from the beginning of the sentence up to word $w-1$ that contains a partially matched rule whose next element can be the root of a tree with $R$ on its left frontier. This is done by computing at each position in the sentence the set of non-terminals that can follow partial derivations of the sentence up to that point.

While this style of prediction works well for CFG's, simple extension of that method to unification grammars founders due to the size of the prediction tables required, since a separate entry in the prediction tables needs to be made not just for each major category, but also for every distinct set of feature values that occurs in the grammar. One alternative is to do only partial prediction, ignoring some or all of the feature values. (The equivalent for CFG's would be predicting on the basis of sets of non-terminals.) This reduces the size of the prediction tables at the cost of reducing the selectivity of the predictions and thus relatively increasing the space that will have to be searched while parsing.

The amount of selectivity available from prediction based on particular sets of feature values depends, of course, on the structure of the particular grammar. In the BBN Delphi system, we found that prediction based on major category only, ignoring all feature values, was only very weakly selective, since each major category predicted almost the full set of possible categories. Thus, it was important to make the prediction sensitive to at least some feature values. We achieved the same effect by splitting certain categories based on the values of key features and on context of applicability. Prediction by categories in this adjusted grammar did significantly reduce the search space.

For example, our former grammar used the single category symbol S to represent both matrix (or root) clauses and subordinate clauses. This had the effect on prediction that any context which could predict a sub-clause ended up predicting both S itself and everything that could begin an S, which meant in practice almost all the categories in the grammar. By dividing the S category into two, called ROOT-S and S, we were able to limit such promiscuous prediction. Furthermore, the distinction between root and non-root clauses is well established in the linguistic literature (see e.g. Emonds [2]). Having this distinction encoded via separate category symbols, rather than through subfeatures, allows us to more easily separate out the phenomena that distinguish the two types of clause. For example, root clauses express direct questions, which are signalled by subject-aux inversion ("Is that a non-stop flight?") while subordinate clauses express indirect questions, which are signalled by "if" or "whether" ("I wonder if/whether that is a non-stop flight.") Thus it seems that the distinction needed for predictive precision at least in this case was also one of more general linguistic usefulness.

A further major gain in predictive power occurred when we made linguistic trace information useable for prediction. The presence of a trace in the current left context was used to control whether or not the prediction of one category would have the effect of predicting another, with the result of avoiding the needless exploration of parse paths involving traces in contexts where they were not available.

Like the rule-groups described earlier, prediction brings to a bottom-up, unification parser a kind of procedural efficiency that is common in other parsing formalisms, where information from the left context cuts down the space of possibilities to be explored. Note that this is not always an advantage; for parsing fragmentary or ill-formed input, one might like to be able to turn prediction off and revert to a full, bottom-up parse, in order to work with elements that are not consistent with their left context. However, it is easy to parameterize this kind of predictive control for a unification parser, so as to benefit from the additional speed in the typical case, but also to be able to explore the full range of possibilities when necessary.

### 3.2 Non-Unification Computations

We are also exploring the integration of non-unification computations into the parser, where these can still provide the order-independence and other theoretical advantages of unification. There are some subproblems that need to be solved during parsing that do not seem well-suited to unification, but for which there are established, efficient solutions. We have built into our parser an "escape" mechanism that allows these limited problems to be solved by some other computational scheme, with the results then put into a form that can fit into the continuing unification parsing. While our work in this area is still at an early stage, the following are the kinds of uses we intend to make of this mechanism.

For example, there is a semantic "indexing" problem that arises in parsing PP attachment, which involves accessing the elements of a relation that can be schematized as (MATRIX-CLASS PREP MODIFIER-CLASS), for example, that "flights" can be "from" a "city". This becomes an indexing problem because different elements of the relation can be unknown in different circumstances. Either class, for example, can be unknown in questions or anaphors. While unification can certainly handle such bidirectionality, it may not do so efficiently, since it will typically do a linear search based on a particular argument order, while efficiency concerns, on the other hand, would suggest a doubly or even triply indexed data structure, that could quickly return the matching elements, regardless of the known subset.

Another example, also semantic in nature, is the task of computing taxonomic relations. There are various ways of encoding such information in unification terms, including one devised by Stallard [7] that is used in the Delphi system. However, that approach is limited in expressivity in that disjointness can only be expressed between siblings and it also suffers from practical problems due to the size of the encoding structures and to the fact that any change in the taxonomy must be reflected in each separate type specifier. Thus there are many reasons to believe that it might be better to replace unification for computing taxonomic relations with another known method, given that it is not difficult to reformulate the answers from such a component to fit the ongoing unification process.

The computations described here are all cases that could be implemented directly in terms of unification, but expanding the unification framework in these cases to allow direct use of other computational approaches for these limited subproblems should be able to significantly increase efficiency without threatening the desireable features of the framework.

## 4 Conclusions

We have described a number of approaches toward increasing the efficiency of a unification-based parser. A compilation step can merge common rule elements automatically, while a limited but efficient form of disjunction allows the grammar writer to combine rules by hand. Prediction using a tuned set of categories can cut down the search space without excessive overhead, and the use of non-unification computation for certain subproblems can add further efficiency. Careful design of this sort, exploring the tradeoffs between the purely declarative and the more procedural, can help us build fast and usable unification-based NL systems without compromising the theoretical elegance and flexibility of the original formalism.

## Acknowledgements

The work reported here was supported by the Advanced Research Projects Agency and was monitored by the Office of Naval Research under Contract No. N00014-89-C-0008. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States Government.

## References

[1] R. Bobrow, Robert Ingria, and David Stallard. Syntactic and Semantic Knowledge in a Unification Grammar. *Proceedings of the June 1990 DARPA Speech and Natural Language Workshop*, to appear.

[2] Emonds, J. E. *A Transformational Approach to English Syntax: Root, Structure-Preserving, and Local Transformations*, Academic Press, New York, 1976.

[3] Susan L. Graham, Michael A. Harrison, and Walter L. Ruzzo. An Improved Context-Free Recognizer. *ACM Transactions on Programming Languages and Systems* 2 (1980), 415–462.

[4] Lauri Karttunen. Features and Values. *Proceedings of the International Conference on Computational Linguistics* 1984, 28–33.

[5] Robert T. Kasper. A Unification Method for Disjunctive Feature Descriptions. *Proceedings of the Association for Computational Linguistics* 1987, 235–242.

[6] F. C. N. Pereira and D. H. D. Warren. Definite Clause Grammars for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* 13 (1980), 231–278.

[7] David Stallard. Unification-Based Semantic Interpretation in the BBN Spoken Language System. *Speech and Natural Language, Proceedings of the October 1989 DARPA Workshop*, Morgan Kaufmann, 39-46.